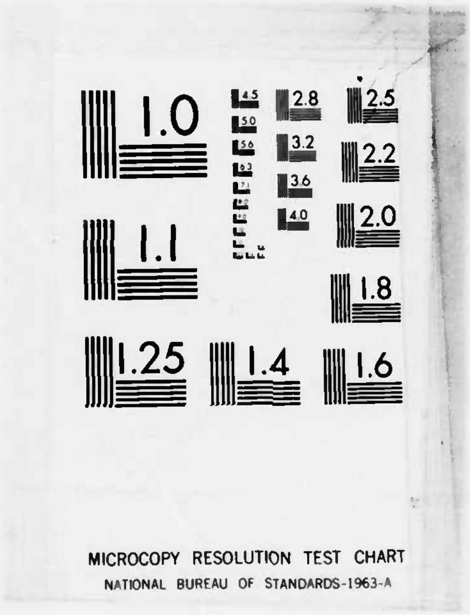
END


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MODEL STATEMENT LANGUAGE/ANALYZER (MSL/MSA): A TOP-DOWN PROBLEM STATEMENT LANGUAGE/ANALYZER (PSL/PSA) APPROACH FOR THE USER DIALOGUE IN DECISION SUPPORT SYSTEMS

by

John Joseph Troy

June 1983

Thesis Advisor: D. R. Dolk

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A132	3. RECIPIENT'S CATALOG NUMBER 273
4. TITLE (and Subtitle) Model Statement Language/Analyzer (MSL/MSA): A Top-Down Problem Statement Language/Analyzer (PSL/PSA) Approach for the User Dialogue in Decision Support Systems		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1983
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John Joseph Troy		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1983
		13. NUMBER OF PAGES 57
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Decision Support System
Knowledge-Based Model Management System
Problem Statement Language
Problem Statement Analyzer
Model Statement Language
Model Statement Analyzer

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis examines the concept of a top-down approach to providing natural language interface and more simple and flexible model manipulation for users of decision support systems, through the use of a model statement language/model

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(20. ABSTRACT Continued)

statement analyzer (MSL/MSA). Patterned after problem statement language/problem statement analyzer (PSL/PSA), the MSL/MSA is a software tool which interprets user input and assists the user through iterative and interactive heuristic problem-solving searches through levels and categories of models, functions, and equations.

Accession For		
NTIS GRA&I	✓	
DTIC TAB	☐	
Unannounced	☐	
Justification		
By		
Distribution/		
Availability Codes		
Dist	Avail and/or Special	
A		

Model Statement Language/Analyzer (MSL/MSA): A Top-Down Problem Statement Language/Analyzer (PSL/PSA) Approach for the User Dialogue in Decision Support Systems

by

John Joseph Troy
Lieutenant, United States Navy
B.S., Marquette University, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
June 1983

Author: John J. Troy

Approved by: Daniel R. Dolk
Thesis Advisor

[illegible]
Second Reader

Richard S. Elster
Chairman, Department of Administrative Sciences

[illegible]
Dean of Information and Policy Sciences

ABSTRACT

This thesis examines the concept of a top-down approach to providing natural language interface and more simple and flexible model manipulation for users of decision support systems, through the use of a model statement language/model statement analyzer (MSL/MSA). Patterned after problem statement language/problem statement analyzer (PSL/PSA), the MSL/MSA is a software tool which interprets user input and assists the user through iterative and interactive heuristic problem-solving searches through levels and categories of models, functions, and equations.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

This thesis investigates the use of a systems analysis and design tool within a modeling or decision support environment with particular attention to the development of a specific modeling tool patterned after existing systems analysis techniques. The specific modeling tool that will be developed will be referred to as a "model statement language" (MSL) and a "model statement analyzer" (MSA). This concept will be discussed briefly below and specifically in Chapter III. The existing systems analysis and design technique that correlates to this is called the "problem statement language" (PSL) and "problem statement analyzer" (PSA), which will be described in detail in Chapter II.

The correlation of a PSL/PSA with an MSL/MSA is indirect due to the nature of the inherent difference that exists between systems analysis and design and decision support environments. Systems analysis and design techniques are structured, top-down views of development whereas modeling software for decision support systems supports unstructured decision-making and has evolved from the bottom-up. For years there have been systems with strong computational algorithms or excellent data access routines whose effectiveness was limited because they were difficult to use. Decision support systems require dialogue aimed at supporting the less technical user. Current state-of-the-art languages evolved from the bottom-up and are directed at the model builder. The need for user-friendly dialog directed at the less technical user justifies an investigation of top-down techniques for decision support system language development.

In order to introduce the notion of MSL/MSA, it is necessary to begin with a background of decision support systems (DSS). A DSS has been defined as an interactive computer-based system that helps decision makers utilize data and models to solve unstructured problems [Sprague & Carlson, 1982]. Although there are different DSS architectures, the basic concept of interaction within a DSS is, from a macroscopic perspective, one where the user interacts with the DSS through a dialogue component:

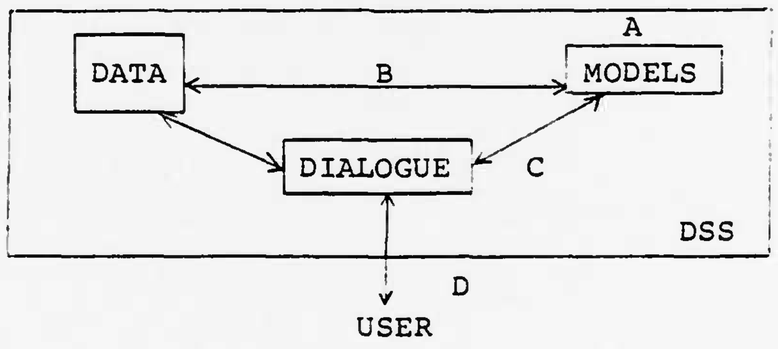


Figure 1. The Dialogue-Data-Models Paradigm

where the elements A), B), C) are labelled as follows:

A) the modeling capability (the model base and its management);

B) which is integrated with the data base (the data-model link);

C) which is invoked using dialogue (the dialogue-model link). [Sprague & Carlson, 1982]

The link labelled "D" in Figure 1 is the basis for this research effort, namely, the 'user-dialogue link'. Much of the power, flexibility and usability characteristics of a DSS derive from capabilities in the interaction between the system and the user. In fact, from the DSS users' point of view, the dialogue is the system. All the capabilities of the system must be articulated and implemented through the dialogue.

One of the objectives of an MSL/MSA is to provide the user a higher-level interface with the DSS models. The MSA would conceptually be included in the "dialogue" block of Figure 1. The proposed MSL concept is that of a non-procedural language on a level of abstraction just below natural conversational English as will be shown in Chapter II. The MSL/MSA concept allows the user with his problem description to interact with a DSS in a form more amenable to him vice that utilized by a model builder (link "C" in Figure 1).

An important assumption in this research is that the modeling environment (the model base and its management) is "knowledge-based" vice "static". Static modeling systems are defined as those in which knowledge about the model and its domain is embedded in the solution algorithm and therefore inaccessible to the user [Dolk, 1982]. In contrast to static modeling, knowledge-based modeling systems (KBMS) are characterized by a knowledge base containing facts about some piece of the "real world" that is being modeled and a knowledge handling capability that can manipulate these facts to make inferences about the environment being considered.

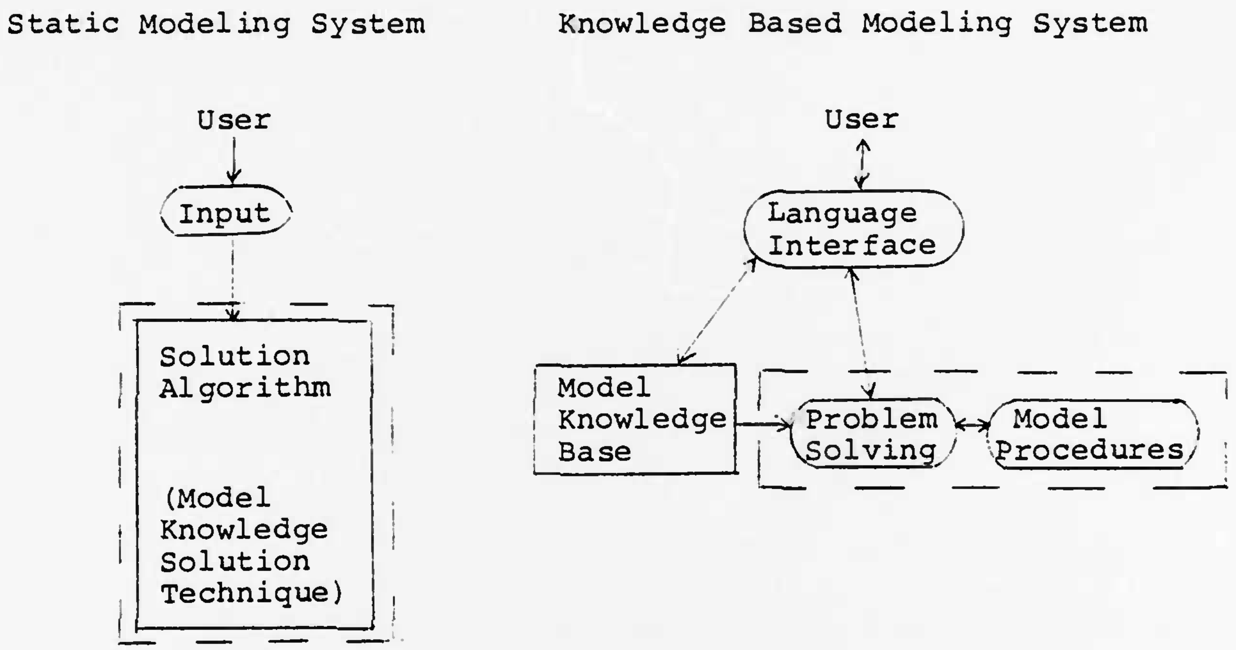
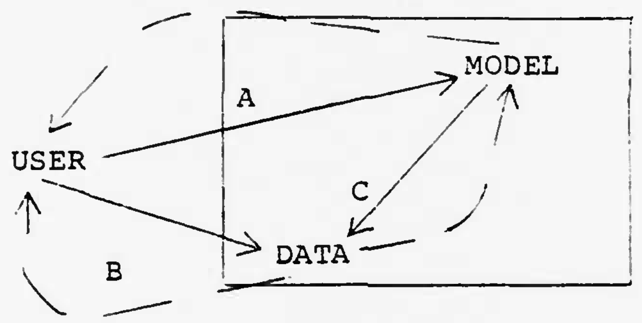


Figure 2. KBMS Partitions Model Knowledge and Problem-Solving. -- Dotted Lines Indicate Entities Transparent to User.

Figure 2 depicts the distinction between the two systems [Dolk, 1982]. The block labelled "language interface" corresponds to the "dialogue" block of Figure 1. Similarly, one of the functional capabilities of the MSL/MSA concept is the ability to interface with knowledge-based modeling systems.

II. LITERATURE SURVEY

The decision maker, or DSS user, is a user of both model(s) and data base(s). Figure 3 shows that software support of a DSS entails the use of languages for the interfaces between the user and the model(s) and data.



Computer Based DSS

A: Language for Directing Computations

B: Language for Directing Retrieval User Language

C: Language for Directing Retrieval Model Language

— →: Response

⟶: Command

Source: Bonczek, Holsapple, Whinston, 1980

Figure 3. Crucial Interfaces in a Decision Support System.

Two of the design criteria proposed for a DSS are:

1) a mechanism whereby models extract data from a data base. A model, then is a user of the data base and must have available some language to direct information retrieval (Figure 3, Arrow C); and

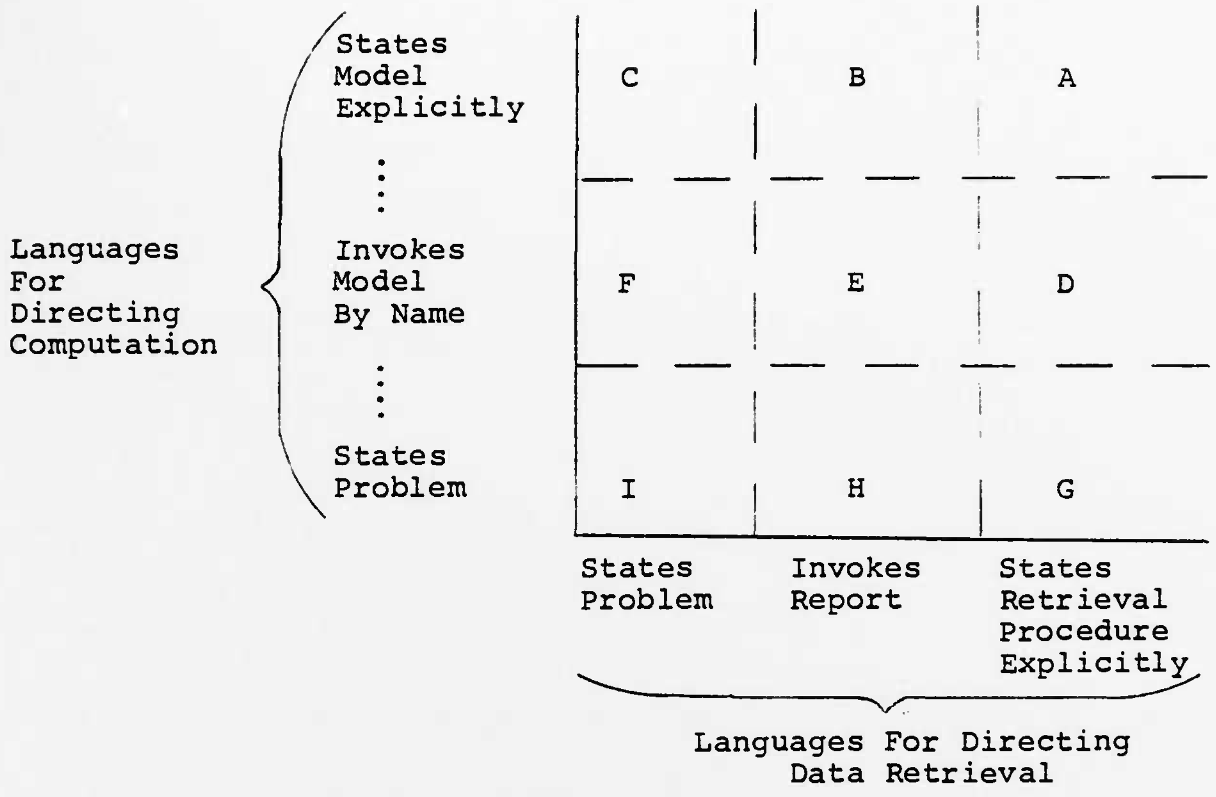
2) a command language allowing convenient, direct access to the data base (Figure 3, Arrow B) and allowing execution of available models (Figure 3, Arrow A). [Sprague and Watson, 1975]

A person using a language to direct (or interact with) a DSS does so with the intention of performing some computation and/or some data management. There are two major categories, then, as seen in Figure (3) that are delineated: languages to direct retrieval and languages to direct computation.

The spectrum of languages used to direct data retrieval has two extremes. One is where the user explicitly states how the data are to be retrieved. At the other extreme is a language where the user merely states the data desired and does not need to know how the data are organized. Similarly, languages to direct computation range between those with which the user explicitly specifies all computations, that is, the user builds the programmed model and those where the user merely states the problem to be solved in terms of the data desired [Bonczek, Holsapple, Whinston, 1980].

Another category that is open to question concerns the ways in which data handling and modeling can be combined into a single system. Figure 4 illustrates a classification scheme for languages.

Languages in category "A" run the gamut from machine and assembly levels to "high level" procedural languages. Common languages in this category are FORTRAN, COBOL, and PL/1. From the user's point of view, retrieval and computation are directed from within a single language. Systems in category



Languages For Directing Computation
States Model Explicitly
Invokes Model By Name
States Problem
C
B
A
F
E
D
I
H
G
States Problem
Invokes Report
States Retrieval Procedure Explicitly
Languages For Directing Data Retrieval

Source: Bonczek, Holsapple, Whinston, 1980

Figure 4. Classification Scheme

"B" access data by invoking a report. Rather than specifying how a particular set of data is to be retrieved for use by the model, the report is automatically generated. An example would be the extended FORTRAN language that allows integrated data base access. Category "C" systems automatically generate code needed for retrieval in response to a statement of the desired data. There is no invocation of a special predefined report generating code, nor is there an explicit statement of how to produce the report. Such systems are especially useful where the types of reports needed by the various models being constructed are unstable, large in number, or unknown in advance of modeling.

Category "D" systems enable the user to direct computations by stating a model's name. This model obtains needed data by explicitly stating retrieval procedures. For example, the user may invoke a simulation model by name, where the model code contains a procedural description of how to retrieve the data it requires. Category "E", as shown in Figure 4, invokes models that acquire data by naming reports. It is useful in situations where reports needed by the various models are static in nature and few in number. Unlike category "D", the retrieval procedures are not intertwined with computational procedures. A predefined group of report types can be produced, each from a specialized report generator that has been preprogrammed. Several models may invoke the same report generator, and each model might call for several specialized reports. Category "F" models which are invoked retrieve data through a language which only states what data are desired. The processor of such a language may be viewed as a generalized report generator. Given a statement of data item types, it determines the logic for producing that report and proceeds to generate the code required to execute that retrieval logic.

Categories "G", "H", and "I" are systems in which the user states a problem in terms of the data desired. Here the respective systems must determine a model (and generate its code) that can produce the desired data. The distinguishing characteristics among categories "G", "H", and "I" is the nature of the language used by an inferred model to accomplish data retrieval. If the model code is punctuated with procedural specifications for data retrieval, then the system falls into category "G". If the model collects data by invoking one or more of a predefined group of specialized report generators, then the system is in category "H". A system in which inferred models access the data base via statements of the data types required would be in category "I".

Bonczek, Holsapple, and Whinston (1980) conclude that existing languages and software systems in the DSS field evolved from those in category "A", are centered in category "E", and will, through research, progress diagonally to category "I".

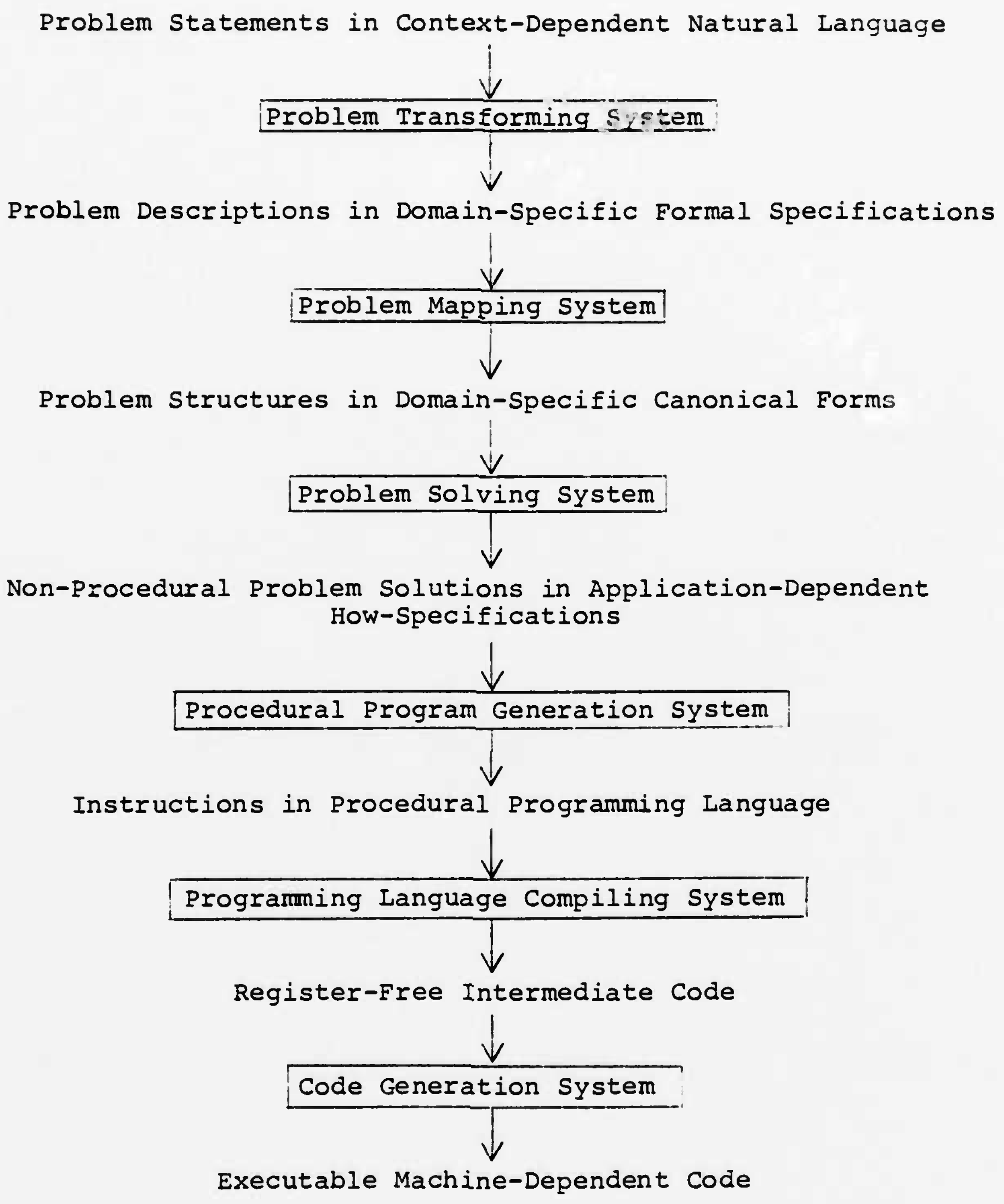
In the area of knowledge-based model management systems, Elam, Henderson, and Miller, (1980) specify four types of information to be contained in the knowledge base, namely, technical, application, language, and model. The <u>technical</u> problem structuring knowledge involves information on mathematical model types (linear programming models, network models, deterministic simulation models, etc.), the parameters that define one model type and distinguish it from other model types (constraints, decision variables, objectives, etc.), and the structural relationships between parameters. The model management system must also have an understanding of the basic applications involved in the problem domain (allocation, scheduling, production, etc.) and the way in which these interrelate. User interface language must be knowledge held by the model management system. Lastly, the model management system must have knowledge about models that have been developed for specific applications (model instances) in order to support the execution and analysis of these models.

The same authors [Elam, Henderson, Miller, 1980], discuss the design of a model management system pointing out first that existing knowledge-based systems utilize either a predicate calculus approach or a graphical approach in representing knowledge. In the predicate calculus approach which relies on "if-then" production rules, there is difficulty in expressing complex concepts and their interrelationships. In analyzing the graphical approach most commonly implemented in the form of a semantic net containing nodes and arcs, they conclude that a major limitation exists in the inability to represent the wide range of conditions that can be easily represented by production rules.

The combination of the two above approaches entails a method described as a "structural inheritance network" ("SI-Net") [Brachman, 1978]. An SI-Net is an expanded graphical representation based on a semantic net which brings together the advantages of graphical and production rule representations. A model management system knowledge base can be thought of as four distinct, but coupled SI-Nets: the technical net, the application net, the language net, and the model net [Elam, Henderson, Miller, 1980]. The technical SI-Net provides information on the type of mathematical programming model and the internal behaviors of a model. The application SI-Net provides an understanding of the basic activities involved in the problem domain and the way in which these activities interrelate. The language SI-Net provides the capability for translating between a user's description of a concept and the system-defined label associated with the same concept. The model SI-Net provides information about the parameters that define one model type and distinguish it from other model types and the structural relationships between parameters.

Wang, and Yu (1983) describe a hierarchical view of DSS software encompassing six transformational processes (or systems) linking seven phases of problem status. As shown in Figure 5, problem statements in natural language are transformed through the hierarchy until solved by machine code.

The problem transforming system transforms a problem representation (problem statement in context-dependent natural language) into another representation (problem descriptions in domain-specific formal specifications). The conversion is of a "what" to another "what" to be further processed by the problem mapping system.


Problem Statements in Context-Dependent Natural Language
Problem Transforming System
Problem Descriptions in Domain-Specific Formal Specifications
Problem Mapping System
Problem Structures in Domain-Specific Canonical Forms
Problem Solving System
Non-Procedural Problem Solutions in Application-Dependent How-Specifications
Procedural Program Generation System
Instructions in Procedural Programming Language
Programming Language Compiling System
Register-Free Intermediate Code
Code Generation System
Executable Machine-Dependent Code

Source: Wang, Yu, 1983

Figure 5. A Hierarchical View of DSS Software

The problem mapping system converts an external problem representation (problem structures in domain-specific formal specifications) into an internal representation (problem descriptions in domain-specific canonical forms). A "canonical form" relates to a simpler and more significant reduction of a problem specification without loss of generality. The problem mapping system uses the "knowledge" of the application area to analyze problem specifications in order to structure the problem into canonical forms which aggregately abstract the same problem. These canonical forms are recognized by the problem solving system. The advantage of canonical forms is implementing the problem solving system without regard to specific applications for which it may be used. The internal representation is still a "what" form which will be solved by the problem solving system in predefined methodology.

The problem solving system is the stage where "what" is converted to "how". The non-procedural problem solution is generated by invoking specialized procedures associated with canonical frames. The canonical form input method can prevent irrelevant knowledge from being accessed and allow canonical frames to interact in specified ways.

The procedural program generation system converts the non-procedural problem solution into instructions in a procedural programming language ready for compilation and execution. Typified by a very high level language such as

APL, the system transforms a "how-specification" to a "do-implementation".

The programming language compiling system performs lexical analyzing, syntax analyzing, intermediate code generation and intermediate code optimization. This system correlates closely to typical compilation processors found in computer science.

The code generation system converts intermediate code into a sequence of machine instructions by deciding on the memory locations for data, selecting code to access each datum, and selecting the registers in which each computation is to be done.

The four SI-Nets of the model management system knowledge base described by Elam, Henderson, and Miller (1980) closely relate to the top four systems of the hierarchical view. The language SI-Net performs the function of the problem transforming system, the application SI-Net performs the function of the problem mapping system, the technical SI-Net performs the function of the problem solving system, and the model SI-Net performs the function of the procedural program generation system.

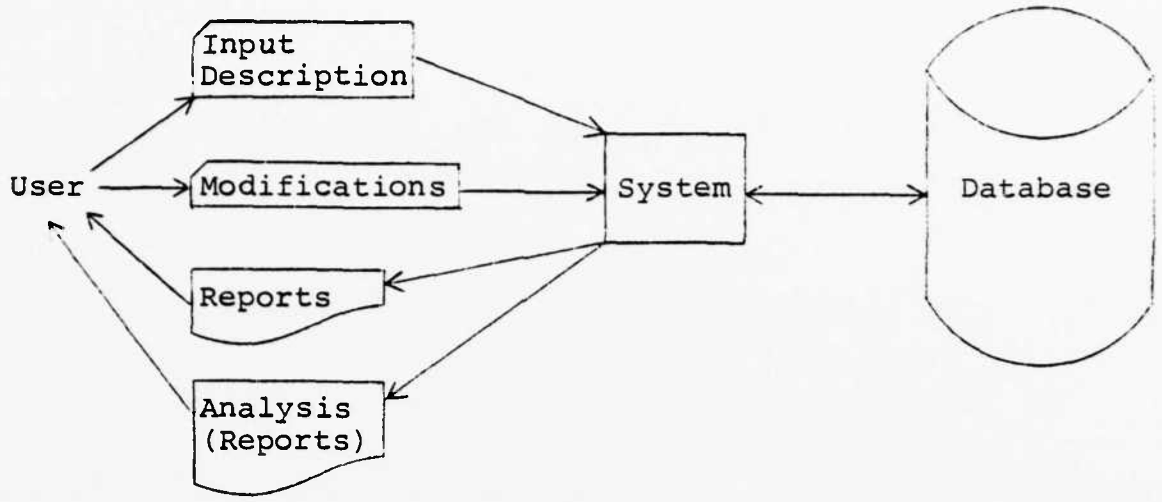
A major conclusion of the authors [Wang, Yu, 1983] in their description of the hierarchical view is that most current DSS software exists in the "procedure program generation system". They also advocate that the incremental development of DSS software will proceed in an upward fashion from "procedure program generation system" through "problem solving system" and "problem mapping system" to "problem transforming system". They contend finally that the evolution will proceed in a manner in which the users of DSS software will be allowed to state problems in less and less specific forms (shifting from "how-to-do" to "what-to-do"). An aim of this research is to examine one vehicle which allows less technically oriented users the ability to state problems in a "what-to-do" format.

All of the literature surveyed above indicates that an indispensable factor of a responsive DSS lies in the interaction between the DSS software and the user. It has also been shown [Bonczek, Holsapple, Whinston, 1980 and Wang, Yu, 1983] that the development of DSS languages will evolve in a "bottom-up" manner which will allow the user to progress from the procedural "how-to-do" to the non-procedural "what-to-do" problem specifications. In this light, it is useful to turn now to an investigation of a "top-down" approach, namely, a systems analysis and design technique utilizing a problem statement language (PSL) and problem statement analyzer (PSA) in order to investigate any feasible applicability to modeling for a decision support environment.

PSL is a nonprocedural language used to define formally the logical structure and requirements of an information processing system. It is nonprocedural in that it facilitates stating "what is to be done" vice "how it is to be done". PSA is a software package which accepts PSL statements as input and generates various documentation and analysis reports. PSL/PSA has been described as a database system for providing a module library for storing source code including a language (PSL) for specifying interfaces in system design which can be checked automatically [Zelkowitz, 1978]. Its major benefits are improved documentation quality and reduced implementation and maintenance cost. The PSL/PSA technique was developed under the ISDOS (Information Systems Design and Optimization System) project which began in 1965 at the University of Michigan [Teichroew, Sayani, 1971].

The basic steps in the life cycle of information systems are: initiation, analysis, design, construction, test, installation, operation, and termination [Teichroew, Hershey, 1977]. PSA produces complete documentation of the requirements of a system, documentation which is intended to be used as input to the design and construction phases of the life cycle [Teichroew, Hershey, Bastarache, 1974]. In summary, the PSA is designed to improve the process of determining requirements for a system by recording, storing, and analyzing the requirements as stated in PSL. A schematic of the PSL/PSA package is shown in Figure 6 below.

PSL is a language for describing existing or proposed systems. The general model on which PSL is based is described by Teichroew and Hershey, 1977. The concept of the model is that a system consists of 'objects' which may have 'properties'. Each property may in turn have 'property



Source: Tremblay, Sorenson, Wig, Perkins, 1980

Figure 6. A Schematic of the PSL/PSA Package

values'. The objects may be connected or interrelated in various ways referred to as 'relationships'. In the specialized model of information systems, the analyst uses PSL to create an object and relate it, via relationship specifications, to the rest of the system in a unit of description called a 'section'. In general, the ordering of sections is not significant, which is in consonance with the nonprocedural aspect of PSL.

Teichroew and Hershey (1977) list eight major aspects of an information system description and PSL contains a number of objects and relationships which allow these major aspects to be described. The eight major aspects allowing all of the information necessary for functional requirements and specifications to be stated in PSL are:

1) System input/output flow;

2) System structure;

3) Data structure;

4) Data derivation;

5) System size and volume;

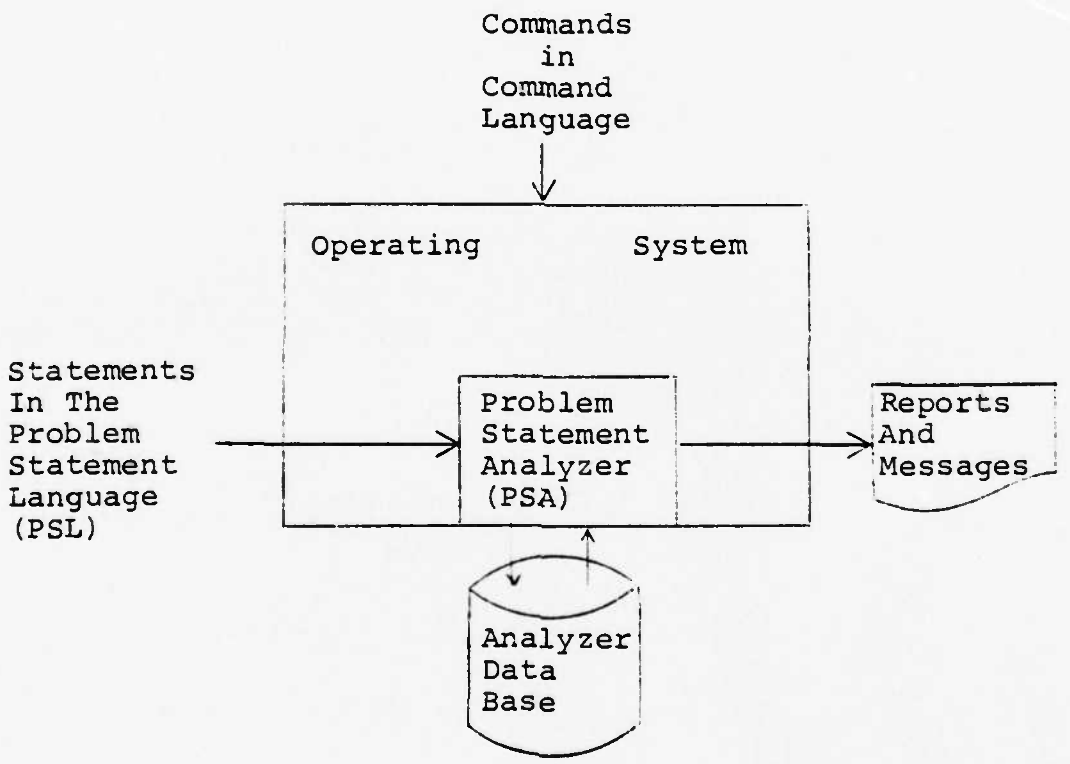
6) System dynamics;

7) System properties;

8) Project management.

Teichroew, Hershey, and Bastarache (1974) delineate the major functions of PSA and its structure. The first function is that PSA provides a facility to compile PSL statements and store an equivalent representation of the problem statement in a data base. Secondly, the PSA contains facilities for the modification of a problem statement stored in a data base. These facilities provide various features including the ability to change the name and type of an object, combine the information stored in two system components and delete the undesired object, input and delete selected PSL statements, and input and delete PSL objects. Finally, the PSA contains facilities to produce documentation from a problem statement stored in a PSA data base and to determine if a problem statement is complete.

These functions are provided by various PSA reports. The primary capabilities of the PSA reports include the ability to selectively print object names contained in a PSA data base, to analyze the completeness of the interactions among process and data descriptions, to display any/all hierarchies contained in the system and/or data, to print a dictionary of the data elements, and to provide a formatted listing of the contents of a data base. Figure 7 below illustrates the general concept of a PSA. It is controlled by the command language of the operating system in which the PSA is embedded.


Commands in Command Language
Operating System
Statements In The Problem Statement Language (PSL)
Problem Statement Analyzer (PSA)
Reports And Messages
Analyzer Data Base

Source: Teichroew, Hershey, 1977

Figure 7. The Problem Statement Analyzer

The various reports may be classified according to their purpose [Teichroew, Hershey, 1977]. The four categories described are as follows:

1) Data Base Modification Reports--A record of existing changes along with diagnostics and warnings used for error correction and recovery;
2) Reference Reports--Information contained in the data base in various formats such as definitions attached to names or properties and relationships for a particular object;
3) Summary Reports--Information in summary form or collated from several different relationships, for example, the structure report which displays selectively some or all hierarchies contained in the system requirements description and the data base summary report which provides project management information by showing the totals of various types of objects and how much has been said about them; and
4) Analysis Reports--Analysis of information in the data base such as the similarities of inputs and outputs, detection of gaps in information flow and unused data objects, and the dynamic behavior of the system.

The approach used with PSL/PSA is iterative, that is, the systems analyst initially describes only a segment of the system. On subsequent occasions, additional information is added to the data base being built during the process until the description is complete. Between iterations, various reports are generated as necessary to perform the analysis required at that stage in the information system description.

Before proceeding to the specifications for an MSL/MSA (Chapter III), it is prudent to review the fundamental objectives of model management systems so that specifications may be analyzed for consistency to the extent that the MSL/MSA applies to the overall DSS. Thirteen fundamental objectives of model management systems are provided [Konsynski, Dolk,1982] as a partial list of attributes characterizing model management systems. This list is provided below:

1. Models should be made available to the decision maker;
2. A wide range of models should be accommodated;
3. The models should reflect the users' (dynamic) world view;
4. The system maintains an indication of model utility and applicability;
5. Simple usage, simple data and model access and recognition;
6. The system should be adaptive to support the users' cognitive style and presentation preference;
7. Reorganization of models should be supported as new world views are perceived by the decision maker;
8. Completeness levels must be satisfied. Selected areas require completeness. Further, we should be able to recognize completeness;
9. The nature of the dialogue with the user must be adaptive and facilitate communication of models to the user;
10. Models should be describable in multiple forms. Translation between various machine and human-convenient forms;

11. The range of users will vary from managers, analysts, programmers, operations personnel and other types of decision makers;

12. A certain degree of validation should be accommodated;

13. Correctness should be established at different levels.

These attributes, along with the top-down approach of PSL/PSA, provide a framework in which to base the development of an MSL/MSA. The MSL/MSA is a subset of an overall DSS and should facilitate the structuring of the user's problem in a nonprocedural "what to do" vice "how to do" procedural manner.

III. IMPLEMENTATION

The three major capabilities of PSA, namely, the facility to compile PSL statements and store equivalent representations, the facility for modifying those statements, and the ability to produce analytical reports based upon stored statements, do have a corollary in the modeling environment. The modeling environment is a higher level of abstraction than PSL/PSA and the greatest direct difference between the PSL/PSA and MSL/MSA environment appears to be in the reports. PSL/PSA produces static documentation while MSL/MSA requires dynamic interactive analysis of the "report" information for continuing user dialogue. Although the modeling environment for a DSS entails additional dynamic functional specifications which will be proposed, the basic top-down approach of PSL/PSA which provides capabilities for storing, modifying and analyzing is assumed to be sound for a modeling or DSS environment.

As described in Chapter I, this research is directed toward the "user-dialogue link" (Figure 1, label "D"). In proposing the specifications for MSL/MSA, the conceptual capabilities may be viewed as a subset of a DSS. Expanding upon Figure 1 under the assumption of knowledge-base, the DSS is depicted as in Figure 8.

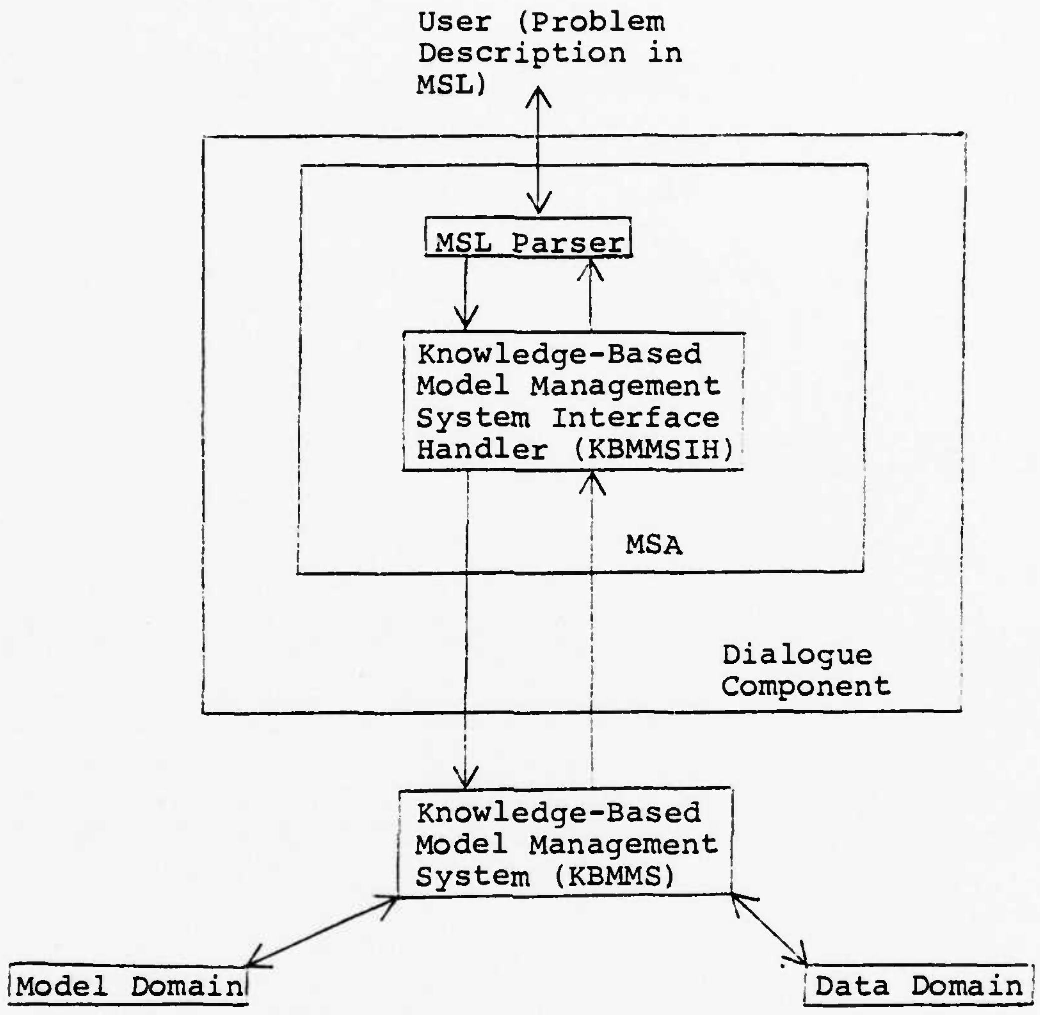

User (Problem Description in MSL)
MSL Parser
Knowledge-Based Model Management System Interface Handler (KBMMSIH)
MSA
Dialogue Component
Knowledge-Based Model Management System (KBMMS)
Model Domain
Data Domain

Figure 8. MSL/MSA Component Interfaces

A. MODEL STATEMENT LANGUAGE SPECIFICATIONS

MSL is a language for describing and interacting with a modeling or DSS environment. A model statement (MS) in MSL can be used to describe a present model, state requirements for creating a model, edit, modify and manipulate models and data as well as to solve a specific application for which the user requires support. MSL is a dynamic, interactive language which allows the user to describe a problem in English-like statements for ultimate solution by the DSS. The user requires little if any knowledge of how a problem will be solved although he needs familiarity with what needs to be solved. The MSL allows the user to concentrate on problem description rather than problem solution. MSL requires little learning by the user because it is English-like. He must be knowledgeable of only minor intricacies of phrase formatting because he can be led through his interaction with the system completely by query. More experienced users may avoid responding to system queries thereby bypassing the iteration of determining which models are applicable to the problem at hand. The more experienced user may know exactly what he wants to do so that a command interface may be more efficient than query mode. The option always exists, however, to default to the query mode at any stage of problem description.

As shown (Chapter II) PSL is based upon a system consisting of 'objects' which may have 'properties' each of which, in turn, may have 'property values'. Objects may be connected or related in various ways referred to as 'relationships'. MSL is based upon a similar system where objects are equivalent to models and properties are equivalent to model parameters. Property values are particular model instances to be solved. Models have relationships to models with respect to their model parameters. For example, a linear programming optimization model is an object with the properties objective functions and constraint equations. Property values are the particular instances associated with a particular property, such as a problem to optimize profit which consists of a particular objective function and constraint equations. Models are related to the extent that their model parameters (properties) are related.

B. MODEL STATEMENT ANALYZER SPECIFICATIONS

The MSA consists of an MSL Parser and KBMMSIH.

1. <u>MSL Parser</u>

The functions of the MSL Parser are to:

1) accept, validate and store user inputs in MSL;
2) query the user for initial information about the problem;
3) provide error messages and resolve grammatical conflict with the user about input statements;
4) compile input from the user for use by the KBMMSIH; and
5) translate requests/information from the KBMMSIH to MSL for interaction with the user.

User-system dialogue is accomplished through a "working" data base called an "MSL Parser". The MSL Parser is a data base of English terminology considered 'working' in the sense that it stores and analyzes MSL input from the user for completeness and consistency as well as transforming such input into the form necessary for use by the knowledge-based model management system interface handler (KBMMSIH). As shown in Figure 8 the parser also transforms information/requests from the KBMMSIH to the user. The parser interacts with the user primarily via query.

The input structure the parser queries from the user follows a most general to most specific methodology. The system first interrogates to possibly determine the model. The queries here are intended to gain any information from the user as to the class of models the problem may deal with, such as linear programming.

The subsequent sections investigate model parameters, parameter values and relationships between models. It should be noted that the structural methodology of querying a user in a top-down fashion is not key for system solution but rather, ordering of sections is intended to be more user-friendly since many users may prefer being led through interaction in a more "logical" way. The parser attempts to complete all sections as in an initial conversation with the user, providing error messages and resolving any input difficulties, prior to compiling the initial section information in a form amenable for analysis by the KBMMSIH.

The MSL Parser is an intermediary between the user and the KBMMSIH. It accepts, analyzes and stores model instance information from the user in a working database. Utilizing an English-like query language, the MSL Parser is a vehicle for interrogation for additional information as requested by the KBMMSIH. Upon solution, the parser provides this information from the KBMMSIH to the user. The parser data base has an extensive vocabulary and is capable of accepting and manipulating user-defined terminology.

Relative to the hierarchy proposed in Figure 5, Chapter II, the MSL Parser corresponds to the 'problem transforming system'. Transforming the user's problem description in MSL statements into the sectional structure for use by the KBMMSIH, the parser converts a "what" (problem statements in context-dependent natural language) into another "what" (problem description in domain-specific formal specifications). The MSL Parser fulfills the functional requirements described by the problem transforming system because it inputs to the KBMMSIH a precise description, without language ambiguity, of what the user wants the system to do.

2. <u>Knowledge Based Model Management System Interface Handler (KBMMSIH)</u>

The functions of the KBMMSIH are to:

1) accept initial information from the MSL Parser;
2) interact with the KBMMS in order to match existing model(s) to the user-defined problem description model instance;
2) iteratively interact with the user through the MSL Parser until model-matching is complete or the user's required model instance is generated;
4) invoke solution of the model instance; and
5) inform the user of the solution.

The KBMMSIH accepts input from the user via the parser and analyzes the sectional information for solution. The KBMMSIH interacts with the knowledge-based model management system (KBMMS) in a search mode attempting to discover existing or potentially existing models in the KBMMS modeling domain which are applicable to the user's current problem. The KBMMSIH iteratively seeks to determine the model category specific to a user's need. This is accomplished by a method of "model-matching". This top-down approach to problem solving utilizes model, model parameter, and relationship information to iteratively narrow down to one model instance. Once the one model instance is discovered or developed, problem solution is performed by the KBMMS. The higher level of problem solving performed by the KBMMSIH, model-matching, is a major functional specification of a MSA.

Figure 9 depicts the overall methodology employed by the dialogue component which includes the MSA.

$$\begin{tikzcd}
\text{Model Description} \arrow[d] \\
\text{Model Comparison} \arrow[d] \\
\text{Model Refinement} \arrow[d] \\
\text{Model Specification} \arrow[d] \\
\text{Model Instantiation} \arrow[d] \\
\text{Model Solution} \arrow[d] \\
\text{Model Report}
\end{tikzcd}$$

Figure 9. Dialogue Component Methodology

Model description is provided by MSL while model comparison and refinement is accomplished by the KBMMSIH which utilizes a catalogue directory and a top down search algorithm described below. Model specification and instantiation derives from the user via the dialogue management facility within the KBMMSIH. Once the model instance which satisfies the user's problem is developed in a form ready for solution, the KBMMSIH invokes the catalogue directory to utilize the corresponding procedure from the procedure library for model solution. This result is reported to the user in MSL via the dialogue capability in the KBMMSIH.

C. KNOWLEDGE-BASED MODEL MANAGEMENT SYSTEM (KBMMS)

The major components of the KBMMS are data bases, model bases, an algorithm inferencing solution, and a catalogue directory. The data bases are associated with model bases of vocabulary, models, and parameters. The algorithm inferencing solution will be described below. A catalogue directory contains a library for generic models, model instances, model names, generic parameter names, parameter instances, MSL vocabulary, and solution procedures, as well as all of the associated data bases.

A key feature of models which reside in the model domain of the KBMMS is that the model base is hierarchically structured. Figure 10 illustrates this hierarchical tree structure which consists of various nodes existing on different levels.

At the very top node, level 0, the compiled sectional information obtained by the MSL Parser in the initial

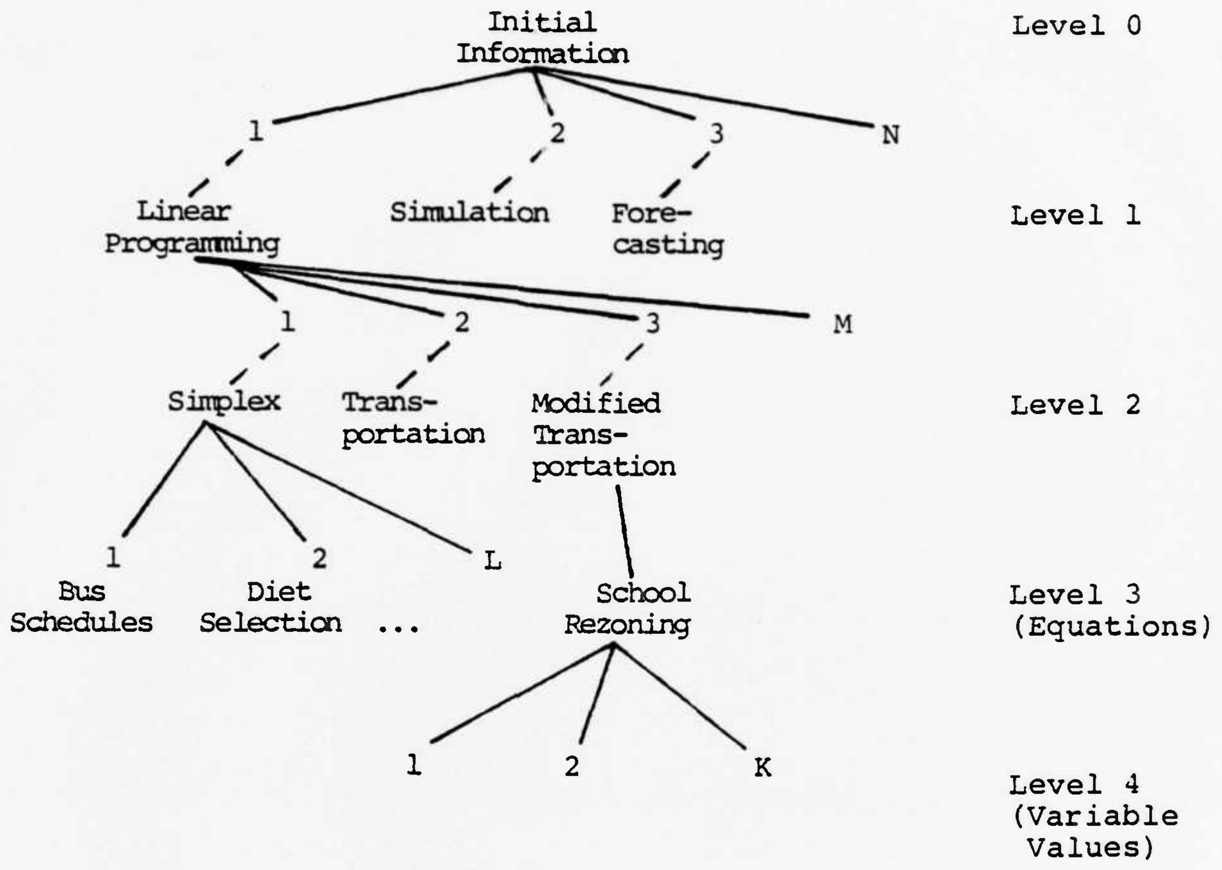


Figure 10. Model Base Hierarchy

conversation exists in the form usable by the KBMMSIH. Level 1 depicts nodes from 1 to N which correspond to different general model classes such as linear programming, simulation, forecasting, and other non-linear programming. Each of these nodes, in turn, branch into a sub-level. For the purpose of illustrating an example, node 1 on level 1 represents the general model type linear programming. It branches to form nodes from 1 to M which are categories of linear programming such as simplex, transportation, modified transportation, etc. Each of these nodes on level 2 expands having branches with nodes on level 3. For example, if node 1 on level 2 represents simplex, then nodes on level 3 represent specific categories of simplex such as bus schedules, diet selection, and a host of other specific applications. Level 3 branches into level 4 which are actual parameter instantiations used to solve a particular problem.

A model-matching algorithm is used by the KBMMSIH to determine if an existing model in the modeling domain is capable of satisfying the solution of a user's problem or if a specific model instance can be developed for the specific problem solution. The algorithms use knowledge about models, model parameters and model relationships obtained from the user either completely from the initial sectional information queried for by the parser, or by specialized request to the user from the KBMMSIH via the MSL Parser. These special requests are invoked when the system is creating a new user-defined model instance or attempting to gain more information from the user because some information in the problem description is either incomplete or not specified. The KBMMSIH attempts to exhaust all possible existing connections in model-matching prior to creating new problem solving model instances.

The KBMMSIH utilizes an artificial intelligence technique for pruning branches of the hierarchical tree structure in order to match the model and ultimately the solution procedure specific to a user's problem. The technique is known as a heuristic search method which utilizes a depth-first search and forward reasoning. Heuristic information is provided by the user when he assists in the search by choosing from a menu of nodes one he desires to pursue at the time. A depth-first search is characterized by the expansion of the most recently generated, or deepest, node first. The search follows a path through the tree downward from the start node. The object of forward reasoning is to bring the problem state forward from its initial configuration to one satisfying a goal condition.

The algorithm employed by the KBMMSIH is one which searches level 1 using the initial information as provided by the user upon initial query as previously described. This information at the start node of level 0 relates to general model and parameter information. The KBMMSIH requests the catalogue directory match model and parameter names and description information to any of those model types in existing libraries. The user is then provided a menu of candidate model matches to select from. Subsequently, this process continues onto lower levels until all levels satisfy solution of the problem. If any level in the search cannot be successfully matched, the search defaults to the preceding lowest level and another node is chosen for investigating the possibility of a match to satisfy the problem.

The case in which this iterative process fails to satisfy the user's problem situation is the one where all nodes from 1 to N in level 1 do not contain successful possible solution techniques on their lower levels. The general algorithm described may be compacted in the following form:

1. Search level 1 for candidate models;
2. Provide menu and documentation to user and query the user for a choice of model to pursue--if no preference, and a model not previously searched exists, choose a model; else, no solution possible;
3. Invoke catalogue directory to display menu and documentation about level 2 nodes associated with the chosen level 1 node to user for a choice to pursue--if no preference, and a node not previously searched exists, choose a node; else, go to 2.;
4. Invoke catalogue directory to display menu and documentation about level 3 nodes associated with the chosen level 2 node to user for a choice to pursue--if no preference, and a node not previously searched exists, choose a node; else, go to 3.;
5. Query the user for the level 4 parameter instantiations which are associated with the chosen level 3 node. If successful, invoke catalogue directory to provide corresponding solution procedure from procedure library. If not successful, go to 4.

For an example which walks through functions of MSL/MSA, a transportation linear programming model is introduced followed by a specific problem with a variation. Consider the model for the general transportation problem as presented in Hillier and Lieberman (1974). It is concerned with distributing 'any' commodity from 'any' group of supply centers, called "sources", to 'any' group of receiving centers, called "destinations", in such a way as to minimize total distribution costs. The model has feasible solutions only if supply equals demand. There are numerous applications which have nothing to do with transportation that fit the general transportation problem model. In general, source i, (i = 1,2,...,m) has a supply of S_i units to distribute to the destinations, and destination j (j = 1,2,...,n) has a demand for D_j units to be received from the sources. A basic assumption is that the cost of distributing units from source i to destination j is directly proportional to the number distributed, where C_{ij} denotes the cost per unit distributed. Table I shows the cost and requirements table for the transportation problem.

TABLE I

Cost Per Unit Distributed

		Destination				
		1	2	...	n	Supply
Source	1	c_{11}	c_{12}	...	c_{1n}	s_1
	2	c_{21}	c_{22}	...	c_{2n}	s_2
	⋮	⋮	⋮	...	⋮	⋮
	m	c_{m1}	c_{m2}	...	c_{mn}	s_m
Demand		d_1	d_2	...	d_n	

The linear programming formulation of the general transportation problem is as follows:

$$\text{Min } Z = \sum_i \sum_j c_{ij}x_{ij} \qquad \text{subject to}$$

$$\sum_{j=1}^{n} x_{ij} = s_i \qquad \text{for } i = 1,\ldots,m$$

$$\sum_{i=1}^{m} x_{ij} = d_j \qquad \text{for } j = 1,\ldots,n$$

$$x_{ij} \geq 0$$

where:

z	=	total cost;
i	=	index of sources;
j	=	index of destinations;
x_{ij}	=	no. of units to be distributed from source i to destination j;
c_{ij}	=	unit shipping cost from source i to destination j;
s_i	=	no. of units supplied by source i;
d_j	=	no. of units demanded by destination j.

As mentioned, there are numerous applications which fit the structure, regardless of their physical context. Table II is a mathematical description of a school rezoning model as presented in Hillier and Lieberman. The problem fits the model of the transportation type with the exception of two additional constraints used to ensure racial balance.

TABLE II

Variable	Description
x_{ij}	No. of students in tract i assigned to school j
d_{ij}	Distance from tract i to school j
b_i	No. of black students in tract i
w_i	No. of white students in tract i
s_j	Capacity of school j
t	Parameter such that $.5-t \leq \text{racial balance} \leq .5+t$

$$\min \sum_i \sum_j d_{ij}x_{ij} \quad \text{for} \quad i = 1,\ldots,10; \quad j = 1,2,3$$

$$\text{st} \sum_j x_{ij} = b_i + w_i \quad \text{for} \quad i = 1,\ldots.,10; \; j = 1,2,3,$$

(all students are assigned to schools)

$$\sum_i x_{ij} \leq s_j \quad \text{for} \quad i = 1,\ldots,10; \quad j = 1,2,3$$

(school capacity not exceeded)

$$\sum_i (.5-t-w_i/(b_i+w_i))x_{ij} \leq 0 \quad \text{for } i = 1,\ldots,10; \quad j = 1,2,3$$

$$\sum_j (.5-t-b_i/(b_i+w_i))x_{ij} \leq 0 \quad \text{for } i = 1,\ldots,10; \quad j = 1,2,3$$

(racial balance)

$$x_{ij} \geq 0 \qquad \text{for } i = 1,\ldots,10; \quad j = 1,2,3$$

(nonnegativity)

In the school rezoning problem of Table II, it should be noted that there are ten tracts ($i = 1,\ldots,10$) and three schools ($j = 1,2,3$). This restriction is chosen as a bound for this particular example but clearly the number of tracts and schools is intrinsic to a user's specific situation. Also, it should be noted that the racial balance constraints for this particular problem are above and beyond the need for fitting any requirements to the general transportation type model. In fact, the "supply equals demand" requirement is filled by the equation which ensures that all students are assigned to schools.

Now the reader will be led through a simulated session of the MSL/MSA which will demonstrate the algorithm presented previously. Upon logging on the DSS, the user with the school rezoning problem described is first greeted and questioned by the MSL Parser of the MSA. The parser first attempts to find out if the user has any knowledge of the general model type involved. In this case, the user contends that his problem is categorized as a linear programming problem of the general transportation type. The next line of query by the parser in attempting to fill the initial sectional information relates to model parameters, in this case the objective function and constraint equations. Here the user's vocabulary is stored in the MSL data base as he provides an English language description of the school rezoning problem. The user need not be familiar with the equation format of Table II so long as he is able to describe the problem by responding to the questions presented him. An example of dialogue that may occur at this stage is as follows:

Parser: Your model is LP General Transportation. You should be concerned with distributing commodity(ies) from source(s) to destination(s) in order to minimize distribution costs, with "supply equaling demand". Does this sound correct for your application?

User: Pretty much.

Parser: I do not understand the term "pretty much". Does that mean

1) Yes, exactly;
2) Yes, but not exactly;
3) Maybe; or
4) No, not in any way?

User: 2.

Parser: Do you know how not exactly?

User: No.

Parser: Very well, then what is (are) the commodity(ies) being distributed?

User: Students.

Parser: From what sources do (does) students come?

User: Tracts.

Parser: What is (are) the destination(s) that students from tracts go to?

User: Schools.

Parser: Does the supply of students from tracts equal the demand for students at schools?

User: Yes.

(At this point the parser has filled the minimal requirement for initial information with regards to model parameter information. It now attempts to fill the parameter value section.)

Parser: How many students are there?

User: 3600.

Parser: How many tracts are there?

User: 10.

Parser: How many schools are there?

User: 3

(At this point the initial parameter section is complete and the last of the initial dialogue deals with relationships between parameters.)

Parser: There should be a relationship (a distribution cost) for distributing students from tracts to schools. In what terms (units) is this relationship defined?

User: Miles.

The initial query by the parser is complete at this point because the sections on general model type, model parameter, parameter value, and relationship information is compiled in a form for use by the KBMMSIH. The initial information is not all-encompassing as a description of this user's problem but it does provide initial information for the level 0 node and allows the KBMMSIH to make its initial search for a candidate model or models.

Upon receipt of the compiled sectional information (for the Fig. 10 Level 0 node) obtained from the user by the parser, the KBMMSIH searches the model base via the catalogue directory in accordance with the general model type and parameters provided by the user. Since the user in this case is fairly definitive, that is, he claims to know the general model type and does provide some parameters within that model, the heuristic search confronts the user with a level 1 model match of the generic linear programming model.

Upon matching the generic linear programming model the KBMMSIH invokes the catalogue directory to provide the corresponding documentation about the model to the user. The dialogue function within the KBMMSIH queries the user about the model presented for the purpose of checking lower level branches of the tree for consistency with the user's problem description. Possible dialogue at this stage may appear as follows:

KBMMSIH: Does the documentation appear to be applicable to your problem?

User: Yes.

KBMMSIH: It is possible this is a linear programming problem.

(At this point the KBMMSIH invokes the catalogue directory to present a menu of level 1 nodes in attempting to prune level 1 branches, as shown in Figure 10, further.)

KBMMSIH: The following is a menu of available linear programming methods. Is there a method which may be feasible?

Simplex

Transportation

Modified transportation

.
.
M

User: Modified transportation.

(Had the user been incorrect in his answer, lower levels in the correct tree path would not have sufficed and the search algorithm would default back to level 1, continuing in a quest to check other nodes on level 1 for a path to possible ultimate solution.)

Now documentation about the existing modified transportation problem is presented to the user. The KBMMSIH invokes the catalogue directory to access the appropriate data base associated with modified transportation. Also in this process on this level of tree search, level 2, the KBMMSIH structures the subsequent dialogue allowing the user to define parameter instances in terms of user-MSL vocabulary. These terms are available in the MSL vocabulary library and are stored via the catalogue directory in the specific parameter instance in the parameter instance library. The following dialogue is now possible at this stage:

KBMMSIH: The modified transportation problem is one which is concerned with distributing commodities from sources to destinations in order to minimize distribution costs with "supply equalling demand". Additionally, there are other constraints involved. Is this the situation apparently?

User: Yes.

KBMMSIH: Initially you stated that commodities = students, sources = tracts, and destinations = schools. Is this still correct?

User: Yes.

Now the KBMMSIH algorithm continues with the iteration of providing the user the menu and documentation of level 3 nodes associated with the chosen level 2 modified transportation node. At this stage the user would choose, or the process of elimination of nodes would eventually present, the choice of school rezoning. At this point the user KBMMSIH would query the user for level 3 equations. The objective function and constraint equations as shown in Table II would be drawn from the user via the KBMMSIH dialogue capability. Actual numerical values associated with the given equations as required on level 4 are also obtained from the user. When this data is obtained the KBMMSIH invokes the catalogue directory to call the corresponding procedure from the procedure library to solve the problem. After problem solution is complete the user is informed of the result by the KBMMSIH.

The above example walks through a somewhat typical description of how the MSL/MSA concept leads a user through problem description to problem solution. The system demonstrates some flexibility and allows the user to concentrate on his problem description rather than on how his problem is to be solved. The KBMMSIH, as part of an MSA, performs the functions of the problem mapping and problem solving systems as described by Wang and Yu (Chapter II). The KBMMSIH performs the function of the problem mapping system by using knowledge of the application area to analyze problem specifications in order to structure the problem in a form amenable for model matching. This is a conversion from "what" to "what". Problem solution is performed by the KBMMSIH after the user specifies and instantiates a matched model which is solvable by the KBMMS. This is a conversion from "what" to "how". The MSL/MSA concept as a top-down approach utilizes a nonprocedural language, incorporates iterative methods, and allows for solution of semi-structured and unstructured problems. MSL/MSA allows the less technical user to concentrate on stating what his problem is instead of how it should be solved. In this sense an MSL/MSA conceptually bridges the gap in the user-dialogue link of decision support systems.

IV. SUMMARY AND CONCLUSION

Users of a computer-based DSS interact with the system through a user dialogue. The dialogue can be used for direct data access or for the invocation of models for producing reports or manipulating the data. The system translation of user input, and the internal management of models should be transparent to the user. To the user, the dialogue is the system. The ease with which a user can employ the dialogue largely determines the success of the system.

Unfortunately, most DSS dialogue components today are not "user-friendly". They require considerable knowledge of the model and data base structures and interactions. The problem here is that the historical bottom-up evolution of the design approach to model and data management facilities has resulted in an overly-technical emphasis in DSS interaction. The concept of MSL/MSA is a top-down approach which will require significantly more effort on the part of the modelers and builders, but promises great benefits for DSS users in ease of use and interaction.

The integration of an MSL/MSA into the dialogue management system will provide the user with a non-procedural, near-English language with which the user can input simple requests by identifying the problem, the required report, or the required data. The MSL/MSA translates the request, determines the necessary data, models, and/or computations, and provides the results. Thus the user is relieved of the technical aspects of data and model management and interaction. The user states the 'what' of the problem. The DSS, through an MSL/MSA-incorporating dialogue, determines the 'how' of the solution.

The MSL/MSA performs this function through a series of steps which precede the steps involved in the current process of converting procedural language to machine code. The additional steps provided by the MSL/MSA may appear to have evolved from the concept of SI-Networks, but are more loosely-coupled and distinct entities. Essentially, they operate, first through the MSL, to accept and interpret a user's problem statement and then, through the MSA, to determine the nature of the problem and possible methods for solution. The user can then be presented with a list of appropriate candidate models from which he or she can make a selection. The MSL/MSA provides a menu of options; it does not itself perform the selection, although such a capability may be possible and worth pursuing after further evaluation of the initial MSL/MSA applicability to the improvement of user-dialogue interaction.

An MSL/MSA is similar in concept (although not in purpose) to the PSL/PSA tools used in software development. The functions and structure of PSL/PSA tools can therefore be used as a model for eventual design and construction of an MSL/MSA, with some modification or provision for the 'dynamic' nature which characterizes the MSL/MSA environment. PSL's objects, properties, and property values are the MSL's models, model parameters, and model instances. Models are 'related' through their model parameters.

The MSA employs a KBMMSIH to perform problem mapping functions as well as many of the storage, modification, deletion and retrieval functions for both models and data which for PSL/PSA are report functions which operate on data alone. The extension of dynamic interaction capabilities, unavailable in PSL/PSA, is made possible within MSA by the translation capabilities of the MSL Parser, whereas PSL is a function of the host system's operating system. Thus the operation of models on data and the provision for user interaction in the process, is the 'dynamic' characteristic which distinguishes MSL/MSA from PSL/PSA.

Models are stored within the KBMMS in an hierarchical tree structure upon which a heuristic search is initiated by user input and response to information requests. The user is thereafter a participant in an interactive process whereby nodes at the successive levels of models, solution methods and equations in the hierarchy are examined for promise in moving toward the goal condition. Problem solution may require several iterations of search through node levels.

This iterative and interactive problem-solving method afforded by the MSL/MSA is a means by which the user of a DSS can concentrate on the problem without the complications currently posed by procedural language DSS interfaces. The procedural language and the successively-lower compiling and code generation systems are, in effect, hidden, or made transparent to the user by the top-down development and addition of problem transforming, mapping and solving systems which translate context-dependent natural language of the user dialogue. Thus, the negative impact on the user which has resulted from the technical, bottom-up development of procedural languages such as FORTRAN can be mitigated with the top-down development of MSL/MSA facilities which accept natural language input and user participation in the problem-solving process.

Today the MSL/MSA is a theory which exhibits significant promise toward the improvement and greater acceptance of DSS capabilities. Further research will be necessary to turn these initial concepts into a working design. It is hoped that this research will help to stimulate such future efforts.

LIST OF REFERENCES

Bonczek, Robert H., Holsapple, Clyde W., and Whinston, Andrew B., "The Evolving Roles of Models in Decision Support Systems", Decision Sciences, Vol. 11, No. 2, April 1980, 337-56.

Brachman, Ronald J., "A Structural Paradigm for Representing Knowledge," Report No. 3605, Bolt, Beranek, and Newman, Inc., 1978.

Dolk, Daniel R., "The Uses of Abstraction in Model Management," Ph.D. Dissertation, The University of Arizona, 1982.

Elam, Joyce J., Henderson, John C., and Miller, Louis W., "Model Management Systems: An Approach to Decision Support in Complex Organizations," Technical Report No. 80-08-04, Dept. of Decision Sciences, University of Pennsylvania, 1980.

Hillier, Frederick S., and Lieberman, Gerald J., Introduction to Operations Research, Second Edition, 1974.

Konsynski, Benn Jr., and Dolk, Daniel R., "Knowledge Abstractions in Model Management," University of Arizona, 1982.

Nilsson, Nils J., Principles of Artificial Intelligence, Tioga Publishing Company, Palo Alto, California, 1980.

Sprague, Ralph H. Jr., and Carlson, Eric D., Building Effective Decision Support Systems, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

Sprague, Ralph H. Jr., and Watson, H.J., "MIS Concepts--Part II," Journal of Systems Management, Vol. 26, No. 2, 1975, 35-40.

Teichroew, Daniel, and Hershey, Ernest A. III, "PSL/PSA" A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977, 41-48.

Teichroew, Daniel, Hershey, Ernest A. III, and Bastarache, Michael J., "An Introduction to PSL/PSA," ISDOS Working Paper No. 86, University of Michigan, March 1974.

Teichroew, Daniel, and Sayani, H., "Automation of System Building," Datamation, 1971.

Tremblay, J.P., Sorenson, P.G., Wig, E.D., and Perkins, D.R., "A Survey of Automated Aids for Systems Analysis and Documentation," University of Saskatchewan, Canada, February 1980.

Wang, Michael S.Y., and Yu, Keh-Chiang, "A Hierarchical View of Decision Support System Software," Proceedings of the Sixteenth Hawaii International Conference on System Sciences, Vol. 1, 1983, 482-89.

Zelkowitz, Marvin V., "Perspectives on Software Engineering," Computing Surveys, Vol. 10, No. 2, June 1978, 197-216.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Computer Technology Programs, Code 37 Naval Postgraduate School Monterey, California 93940	1
4.	Associate Professor Daniel R. Dolk Code 54Dk Department of Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
5.	LCDR John R. Hayes, SC, USN, Code 54Ht Department of Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
6.	LT John J. Troy, USN 9727 Lindgren Avenue Sun City, Arizona 85373	3